Elora Newcomb

Instructor: Olga Mill

CS-360 — Mobile Architect & Programming 23EW6

16 October 2023

**Reflection on Implementing Secure Coding Practices and Zero Trust**

We have spent the majority of this course exploring the dynamic field of cybersecurity and the best practices for writing code that is secure against attacks. In retrospect, I am astounded by how much I have learned in this subject, which will surely influence how I write code in the future.

**The Zero Trust Paradigm**

The "zero trust" principle was introduced throughout as a central concept; the conventional security model, where a firewall is the only line of defense and all data within a network is taken as read, is challenged by this method. However, the zero-trust mentality mainly advocates a radical change in outlook, arguing that "no one is safe" and that trust should never be presumed, not even within the boundaries of the company's own network.

In today's digital world, as attacks become more complex and destructive, zero-trust protection is essential: since security is an ongoing activity rather than a one-time effort, zero trust places a premium on vigilance and a strong security posture.

**Impact on Secure Coding**

While the main topic of this module was zero trust, it is important to consider how it relates to safe programming, and how it can be used to identify security vulnerabilities. In general, we shouldn't blindly rely on any part of our codebase, no matter how constantly we improve it.

The importance of safe coding techniques, like those covered in this course, is on the rise. Input validation, memory management, and the correct usage of data structures are all the more important as you move toward zero trust architectures. Secure coding methods lay the groundwork for writing programs that, although cannot be trusted one hundred percent, can be verified as secure.

**Applying Best Practices**

We, as developers, must be conscientious about adopting and enforcing best practices both ethically and morally. We need to be aware of the possible loopholes and security flaws in our code and work to fix them, admitting our mistakes as soon as they are found so they can be fixed right away, minimizing negative effects on the userbase and the system itself.

**Recommendations for a Future Security Policy**

In this course, we've explored the importance of developing a robust security policy for C/C++ that effectively guards against attacks. Proactive prevention and continuous monitoring will be crucial in the future for software security. Therefore, I recommend that our organization consider the following for our security policies:

1. Comprehensive Education: Ensure that all developers are well-versed in secure coding practices and understand how these practices align with the zero-trust approach.

2. Continuous Testing: Implement a comprehensive testing process that includes regular code reviews, automated vulnerability scanning, and penetration testing to detect and fix vulnerabilities as soon as they arise.

3. Security Culture: Promote a security culture within development teams, as well as enable developers to share their insights about potential vulnerabilities.

4. Incident Response: Develop a robust incident response plan in order to respond to security breaches as quickly and effectively as possible. Preparation for incidents is just as crucial as prevention.

5. Keep Abreast of Trends: Stay informed about emerging threats and vulnerabilities. The security landscape is continually evolving, and it's essential to adapt our policies accordingly.

To keep up with the rapid digital world development, we must adopt the zero-trust paradigm in conjunction with secure coding practices. By adhering to these guidelines, we may create software that is robust enough to survive cyber threats as they evolve. As a programmer, I look forward to implementing these lessons in my own programming projects and making a significant contribution to the field of safe, reliable software.